



The Definitive Guide to

Micro Frontends for Mobile

Table of Contents

Introduction	03
Section 01	
How mobile apps are built today	04
Section 02	
The definition of micro frontends	07
Section 03	
Considerations when adopting a micro frontend architecture	10
Section 04	
Best practices for organizing teams when implementing micro frontend architecture	12
Section 05	
Most common use cases	23
Summary	26

Introduction

Mobile application development, whether native or hybrid, has historically come with a variety of challenges, especially in larger organizations. Scaling development, maintaining feature parity across platforms, and constantly deploying bug fixes are just a few of the more common issues. Development teams at large enterprises are nothing if not persistent, so many of these issues have just become part of the mobile application development process.

However, advancements in technology have encouraged development teams and technical architects to rethink application architecture in order to alleviate some of the headaches they face regularly. One such advancement is micro frontend architecture. While it's not a magic bullet, it can be a massive step in the right direction.

This ebook will explore how micro frontend architecture can alleviate some of the pain points typical in large-scale application development. We will cover the definition of micro frontends, advantages and disadvantages of implementing micro frontends in mobile app development, and best practices around adopting micro frontends within your organization.

How mobile apps are built today

Monolithic architecture is the most common way to develop mobile applications. This style of development looks a little something like this:



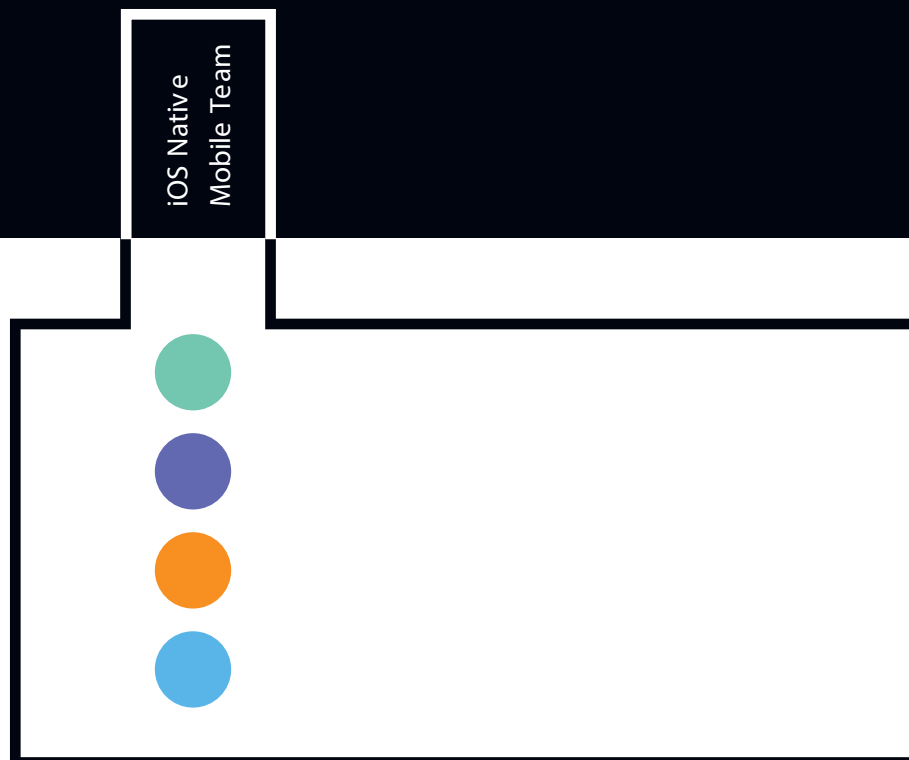
A single team of dozens of engineers handle the different components of building the application, including developing the frontend experience of the application, connecting and building a backend service, and application testing.



The application uses a monolithic architecture, which requires the team to build the application together using a single code base. Whenever the app needs to be updated, the entire stack needs to be compiled and deployed in unison.



The release cycle in a monolithic architecture is therefore inevitably slow, with larger and less frequent updates and narrower pipelines for progressive updates.



While this approach may feel clunky, there are some benefits to keeping the status quo.

Because each component in a monolithic architecture is tightly coupled and tied to one codebase, this usually increases communication and collaboration between developers who get the work done.

Monolithic applications can also benefit smaller development teams and applications in the early stages. With fewer components to worry about and a smaller application footprint, testing can move quickly.

Despite these benefits, developing applications in a monolithic architecture doesn't necessarily align with the goals of IT leaders within the organization.



**You can think of
micro frontends as
micro services for
the user interface.**

Grgur Grisogono

Director of Engineering
Modus Create

Outcomes IT leaders and architects want to see

As demand to curb costs while maintaining or improving customer experience become high priority in enterprise organizations, IT leaders and architects are tasked with finding ways to implement these priorities within their teams by improving the mobile application development process.

Specifically, they're hoping to:

Scale app development quickly

As apps increase in size over time there is a constant battle to keep shipping updates. As teams add more developers, there is additional overhead involved. In a monolithic application, the agility needed to scale app development simply isn't there, since every bug fix and feature update requires a fully synchronized effort from the entire team to package and upload the changes.

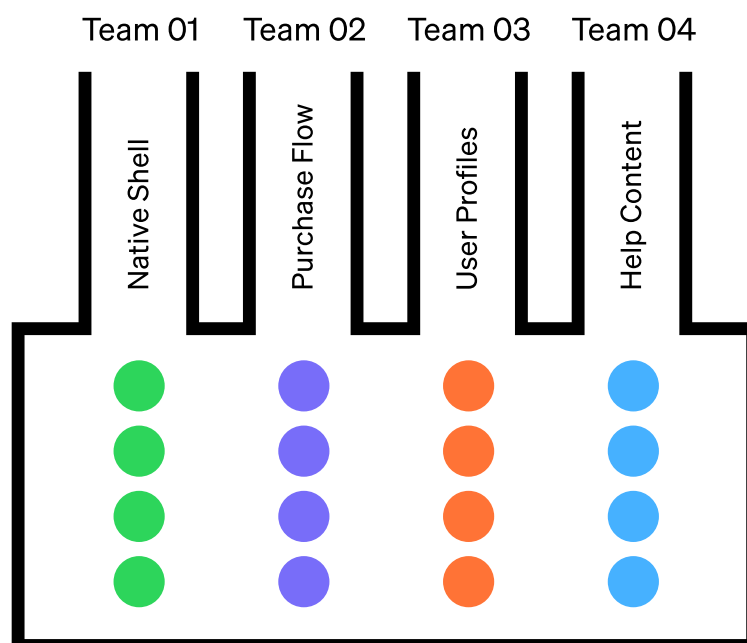
Improve developer efficiency

When developers work together on the same codebase on the same project in a monolithic architecture, they rely on each other to get the work done quickly. When one developer doesn't finish a task on time, the entire team needs to wait until the task is completed and progress is stalled. It also adds complexity to the project, which inevitably leads to a bottleneck effect for development teams.

Maintain a great customer experience

Difficulties with scaling applications and poor developer efficiency can lead to a bad customer experience overall. Bugs need to be fixed as soon as they're found and new features need to be added to keep customers happy and loyal. The slower value is delivered, the more trust will be eroded.

The ideal outcome looks something like this:



The current way to build apps with a monolithic architecture fails to satisfy all of these outcomes in large enterprises.

This is where micro frontends come in.

SECTION 02

The definition of micro frontends

“In short, micro frontends are all about slicing up big and scary things into smaller, more manageable pieces, and then being explicit about the dependencies between them.”

Cam Jackson

Full-stack web developer and
lead consultant at Thoughtworks



Coined by Thoughtworks in 2016, the term micro frontends refers to the concept of breaking up the frontend of an application into different feature slices or segments. Micro frontend architecture is counter to the monolithic architecture approach as it allows different teams to work on different parts of the application simultaneously. Each team can own a given feature end-to-end, which ensures a focused experience for development teams and an overall cohesive experience for customers.

Advantages of micro frontend architecture and problems solved

Improve efficiency and build better apps faster with micro frontend architecture.



Scale app development quickly

Micro frontends can help organizations continue to increase the number of developers working on one product without decreasing the speed to market. They make development fast and efficient by enabling teams to work on their components in parallel without affecting the work of other teams.

Micro frontends increase scalability by making it easier to integrate new features and updates, since each component can be updated independently without impacting the rest of the application.



Improve developer efficiency

Micro frontends let individual teams focus on their own feature while maintaining a higher sense of ownership. Teams can hone in on the most valuable code that they build for their customers.

When organizations scale they start to build teams that have a more specific focus. As micro frontends are usually separated based on feature, each team can focus on their area of expertise.

Micro frontends ultimately reduce complexity among development teams. They allow teams to keep their area of ownership more manageable. This makes it easier for teams to work on their own features in parallel and with their own timelines for releases.

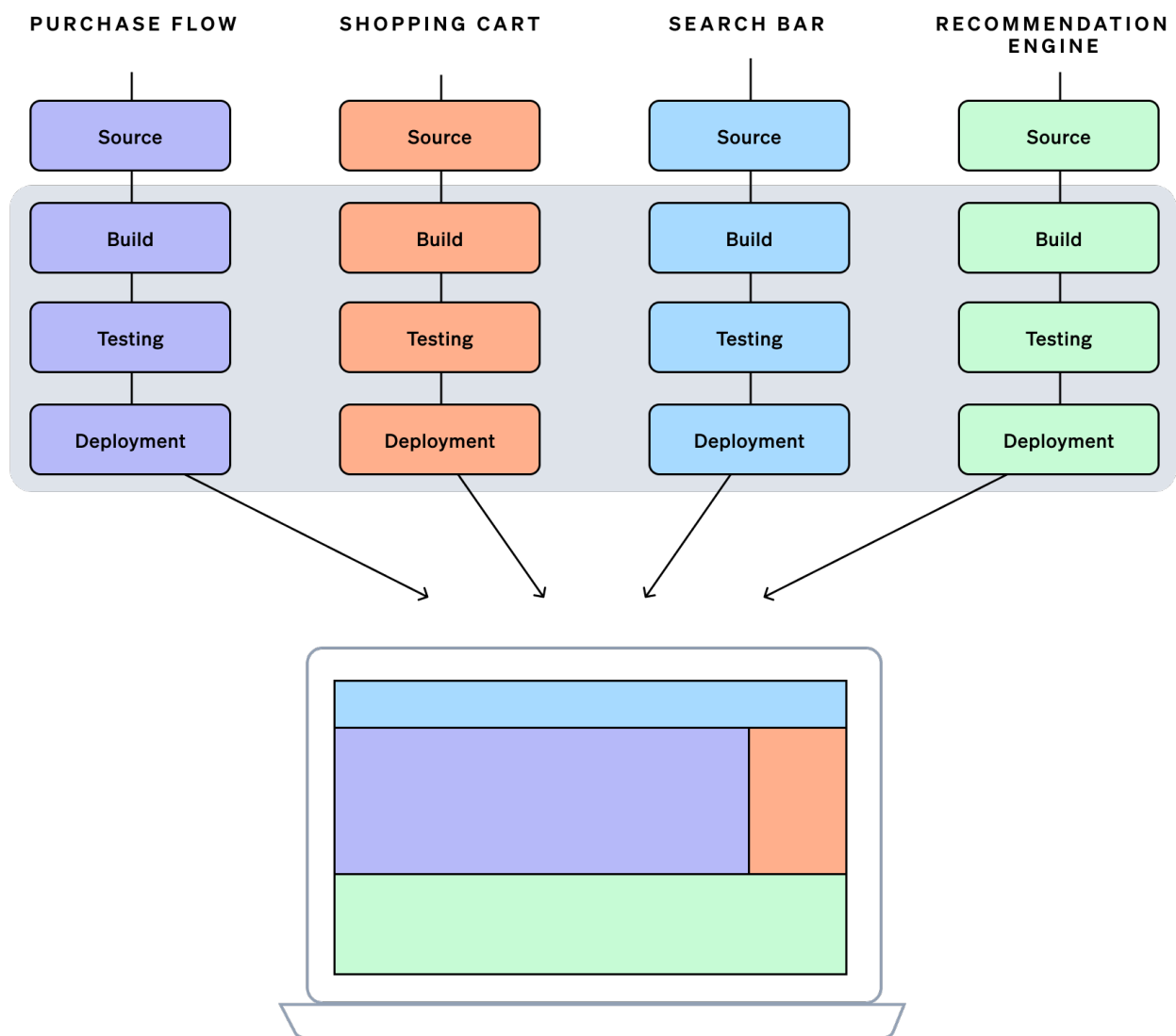


Refine or maintain a great customer experience

Micro frontend architecture empowers organizations to build and maintain customer trust through quickly and efficiently deploying updates and bug fixes. The faster you can deliver value to your customers, the better their overall experience will be.

These problems are solved thanks to web technologies.

When micro frontends are mentioned in the context of application development, it is almost always in the context of web applications. What is it about the web that makes this a great platform for micro frontends? Let's dive into how web technologies have solved these architectural challenges and why they are a good fit for micro frontends.



The web allows for scalability, speed of delivery, and reduced complexity. In this way, it is perfectly positioned to break up a monolithic frontend into different features.

Distributed loosely coupled system

The web is defined by a set of specifications and standards on how content is created and browsers should display that content. As long as you adhere to these conventions, you can design a system that has pieces that operate independently from each other.

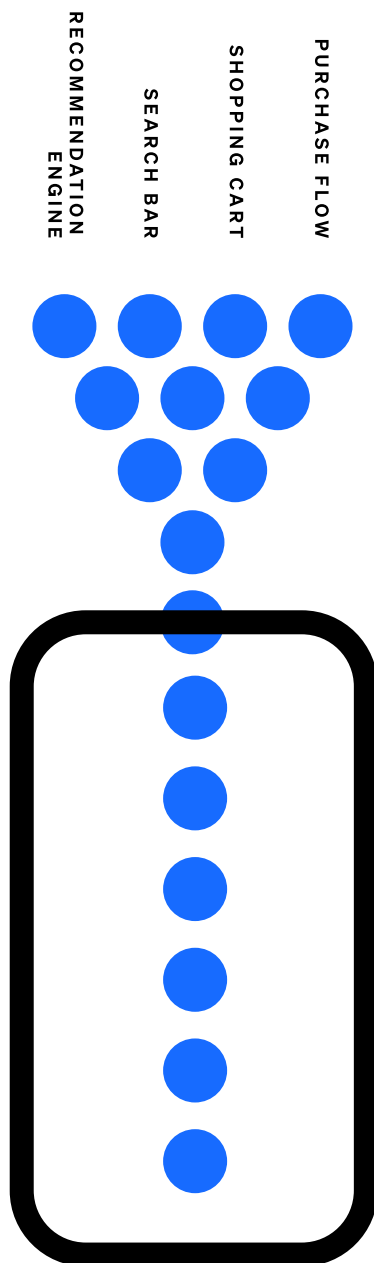
Web code is evaluated at runtime

The JavaScript application code is delivered as source code, then it is compiled, interpreted, and executed. This allows development teams to bring in code from other apps and execute them like they were part of the original application. Web code is downloaded at the time it is needed, so there are no updates or hotfixes required to ensure users are on the latest app version providing them with the best experience possible.

How scaling, developer efficiency, and customer experience are currently handled in native application development

How do native teams make their applications more scalable, improve developer efficiency, and improve customer experiences? There are a few approaches that teams have implemented. Some are more common than others, and this can differ between operating systems. The most common approach is to break them down into a module or plugin-based architecture. During development each team can work on their own specific module. This allows teams to test and develop in isolation, to a point. While it can improve the native development process, there are a few things to consider with this approach.

Disadvantages to breaking native apps into modules



Before breaking down the app into modules, native teams need to ensure the architecture can scale. While each team controls their own module, they need to come together to ensure common functionality meets the needs of each group. Since build times can be lengthy, it's common to use Dev Apps to test and develop each individual module. However, to understand how each module works together, they eventually need to be tested and released together. Therefore, intrateam communication needs to be solid with every release. Overall this could help scale the number of teams but at the cost of developer efficiency.

Within Android you can use the App Bundle feature called Dynamic Feature Modules. This feature allows you to dynamically download specific feature modules rather than the entire application, reducing the install size of the application and improving delivery. However, Dynamic Feature Modules lack the same speed of delivery that we gain with micro frontends. These feature modules are also developed in the same way and coordination difficulties are still prominent. Additionally, development teams cannot independently update and deliver the code within these features modules.

Alternatively, organizations can use Web Views to break down the frontend into feature slices, and we will expand on this later.

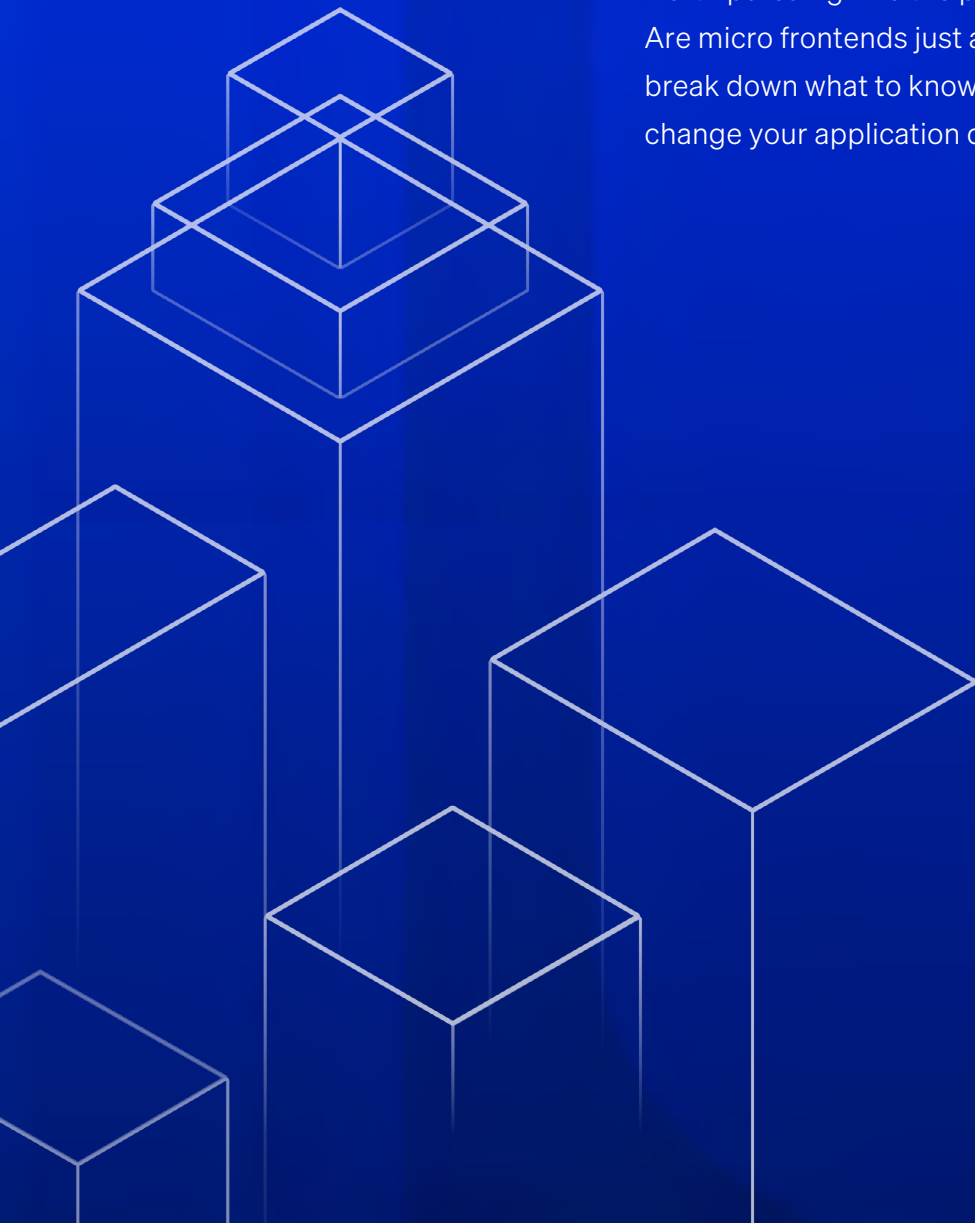
The important thing to understand is that even with these approaches, native apps are still monoliths that use a single code base. The monolith is broken down into modules and then repackaged for every release. Even if breaking down the application into modules speeds up development, application delivery can be delayed due to app store reviews and other bottlenecks during deployment.

Native application development inherently doesn't do a great job of abstracting the benefits of speed of delivery, reduced complexity, and scalability. The amount of work needed to combine at build-time rather than runtime increases complexity, reduces delivery speed due to review processes, and limits scalability since you're managing a monolithic codebase.

SECTION 03

Considerations when adopting a micro frontend architecture

While there are many benefits to using a micro frontend architecture in large-scale enterprise applications, there are some organizational and structural questions to consider. Is the team big enough to make micro frontend architecture worth pursuing? Do the projects have enough resources? Are micro frontends just a trend? The following section will break down what to know before making the decision to change your application development strategy.



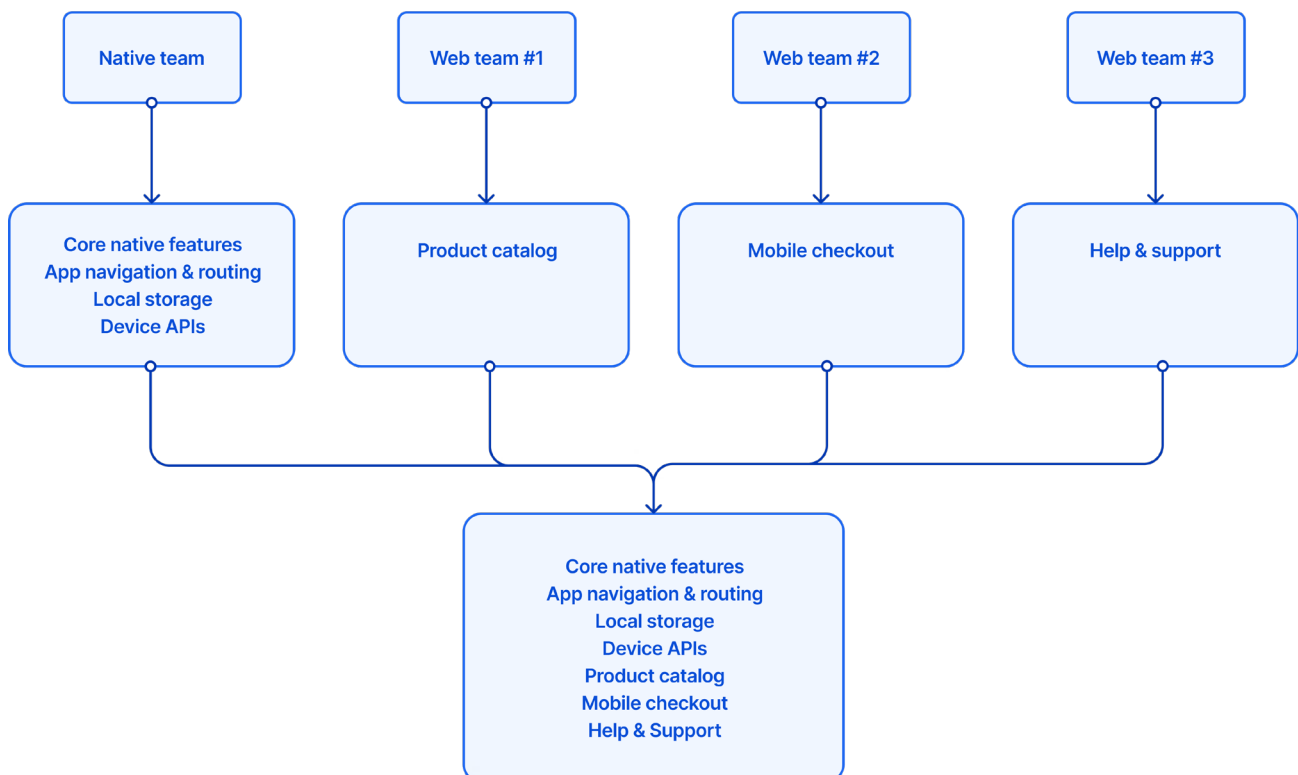
Organizational considerations for a micro frontend architecture

Though implementing a micro frontend architecture is considered a technical decision, it is just as much a team and organizational decision.

Team coordination

How is your team structured today? Teams must first be broken into feature teams, each handling a small section of the application. This requires coordination across many parts of the organization to ensure there are no duplicate efforts or overlaps in project tasks. Dependencies between teams also need to be thought through. Each team manages their own feature, but together, these features need to function as a single application.

Come up with a strategy to determine which teams work on which features, how these features will work together, what component libraries and design systems will be adopted, and what languages will be used across each micro frontend. With these structural decisions made, the project can run smoothly.





Repurposing **web developers** is great and they gain new skills, aiding retention.

Enterprise Architect
Healthcare Industry

Resources for building and maintaining micro frontends

Does your organization have both native teams and web teams? Larger organizations tend to have both in great quantities, so resources for building and maintaining micro frontends will not come at a significant cost. For smaller teams in smaller organizations who work on multiple projects, resources may be constrained because splitting teams up by feature requires focused resources. Team members will not be cross-collaborating on different features, so it is important to know how many team members will be required for a given project.

Scale and scope of building micro frontend applications

Is your project big enough to warrant using a micro frontend architecture? The benefit of a micro frontend architecture is the ability to reuse web code across mobile and web applications—not only within a single mobile application but across multiple applications and experiences. Breaking down an application into features requires that the functionality is robust enough that the project would benefit from multiple developers to build, maintain, and scale them. The scope of the project should also be considered wide enough to cover multiple operating systems and web experiences.

Micro frontends aren't necessarily for everyone.

We understand the excitement around using a micro frontend architecture. For many, it means scaling application development and increasing productivity while maintaining a great customer and developer experience. However, not all organizations stand to gain significant benefits from adopting a micro frontend architecture. If any of the following apply, you may want to reconsider if micro frontends are the best approach for your projects.

You have a small team.

Team size is a major factor in determining if micro frontend architecture is right for your organization. Consider a startup with one native developer and one web developer. The size of the development team is simply too small to function under this architecture. However, when the company is ready to scale, it is worth thinking about migrating from a monolithic to micro frontend architecture before adding new members to the team.

You have a small app.

Much like the size of the team, the size of the application is a major factor in deciding whether or not to adopt micro frontends. If the app is too small to break down into feature slices, consider staying with a monolithic architecture until more features are added.

Your app isn't cross-platform.

If the application will remain solely within the iOS or Android ecosystem without the chance of branching out into the other operating systems, there is no need to use micro frontends until the platform footprint grows.

Just because it's a trend.

To be clear, we don't consider micro frontends to be just a trend. We see the long-term potential and amazing capabilities micro frontends provide to large organizations. However, we understand that micro frontend architecture is not for everyone. Think critically about your organization's ideal outcomes, the problems you're trying to solve, and how micro frontends can help you reach your goals.

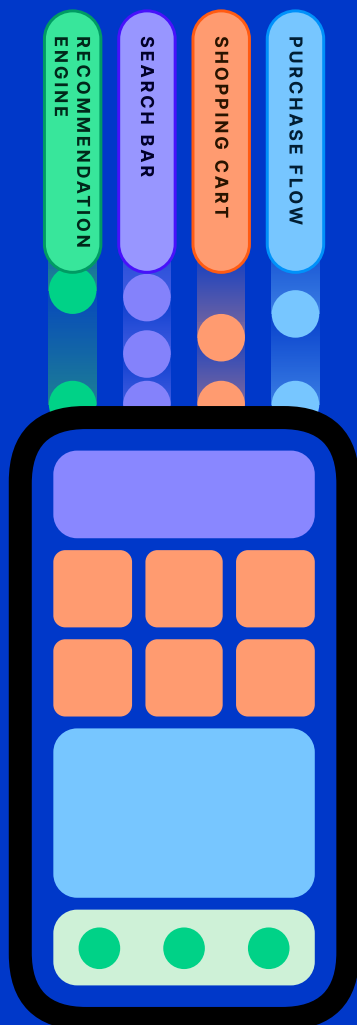
Best practices for organizing teams when implementing micro frontend architecture.

Initial Organization

Organizing by feature team is easiest when the teams are already focused on specific functionality, relying on the technical and domain expertise of team members.

Here are two guidelines we recommend when breaking up an existing application that is currently supported by a single team:

1. Organize the teams based on the expertise/skill sets of the individual members
2. Clearly define the functionality and areas of responsibility for each team



Collaboration

Understand that even though you have separate teams and clear responsibilities, there will still need to be collaboration across teams. Ensuring that common functionality is implemented in accessible ways and dependencies are properly managed are critical for long term success.

The level and the type of collaboration needed will vary from project to project, especially if the shell application is crossing technological boundaries (native code with Web Views) or if the teams are working on a single platform (hybrid web native).

How-to for web native apps

For web native mobile apps using a tool like Capacitor, there are a number of ways to break down the monoliths. This is possible because micro-frontends are a very well established technology for web-based applications.

Module Federation

The most common tool used today for web based micro frontends is Module Federation. This allows applications to create contracts between micro frontends, the single-page application (SPA) that they are used within, and the dependencies that they may share. As long as the contract stays consistent and is not broken, each individual piece can be updated and deployed independently.

This works best when all the applications share common functionality and use the same JavaScript framework, such as Angular or React. By allowing Module Federation to coordinate the dependencies at runtime you can maximize speed and reduce duplication of code.

Federated Capacitor

Federated Capacitor is a tool that allows for Over-the-Air (OTA) updates of both the shell application and of each individual micro-frontend. When Federated Capacitor is used in conjunction with micro frontend tools like Module Federation and single-spa you regain the advantage of independent releases of micro frontends in web native applications.

“[Micro Frontends are] all about bringing back the agility of small teams, even though you have a huge overall software system.”

Manfred Steyer

Trainer, Consultant, Author
Angular Architects

Single-spa

A popular alternative to Module Federation is single-spa. It allows teams to use multiple JavaScript frameworks within the same single page application (SPA), allowing you to split code by functionality and have separate sections written in the framework of choice.

This works best when you are optimizing for independence of developer teams and allowing them to choose their own technology. However, this comes with the downsides of increased lines of code, a larger surface area to maintain, and potentially slower applications.

Setting boundaries

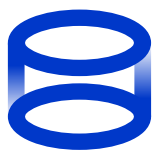
There are quite a few advantages to using micro frontends in web native applications.

With all the code running on the same web platform, you gain flexibility in how to coordinate the primary application and all of the micro frontends that it includes. You can introduce them as multiple views, single components, or even widgets.

You can also adapt your existing testing procedures of web native to cover the micro frontends and the primary application. Commonly, you'd have a series of end-to-end tests that cover the entire experience or integration points from the primary application team's perspective. However, allowing micro frontends teams to have their own test suites works as well.

If you're considering adopting a micro frontend strategy in a single platform (like web native), you must be explicitly clear about the boundaries between your teams. Team members should understand which responsibilities are theirs to ensure seamless collaboration. You must also be clear on when code sharing is acceptable versus not. While micro frontends make code sharing easier than ever, sharing code introduces new potential risks that your team should be aware of.

How-to for native apps



In general, deploying micro frontends for native mobile applications isn't possible today. However, there is a solution that allows you to develop and deploy web-based micro frontends within native mobile applications – [Ionic Portals](#).

[Portals](#) is a tool that allows you to integrate views into a native mobile app using web technologies like HTML, CSS, and JavaScript. This means that you can use Portals to create web-based micro frontends and seamlessly integrate them into your native mobile app. By using enhanced Web Views, Portals allows you to communicate between the native app and the web content for data exchange and event handling.

These micro frontends are located on the mobile device, not remotely on a server, and create a seamless user experience. Each Portal can be remotely configured with OTA updates allowing for autonomous and independent development and deployment. Micro frontend teams aren't required to coordinate a larger release of the native application, saving time and resources.

Considerations

When introducing micro frontends into a native application, the primary consideration is usually how it might affect the user experience. The coordination of multiple teams with different technology platforms can add to this complexity. Thinking about the micro frontends as clearly defined features of a larger native application creates guardrails:

- **Is the feature meant to be an immersive experience?** Then showing the experience within a full screen modal will help keep focus on the task.
- **Is the feature a drop in component?** Then have it displayed as a piece of a larger view experience.
- **Is the feature a core element of the application?** Then you might make it the entire context of a tab stack.

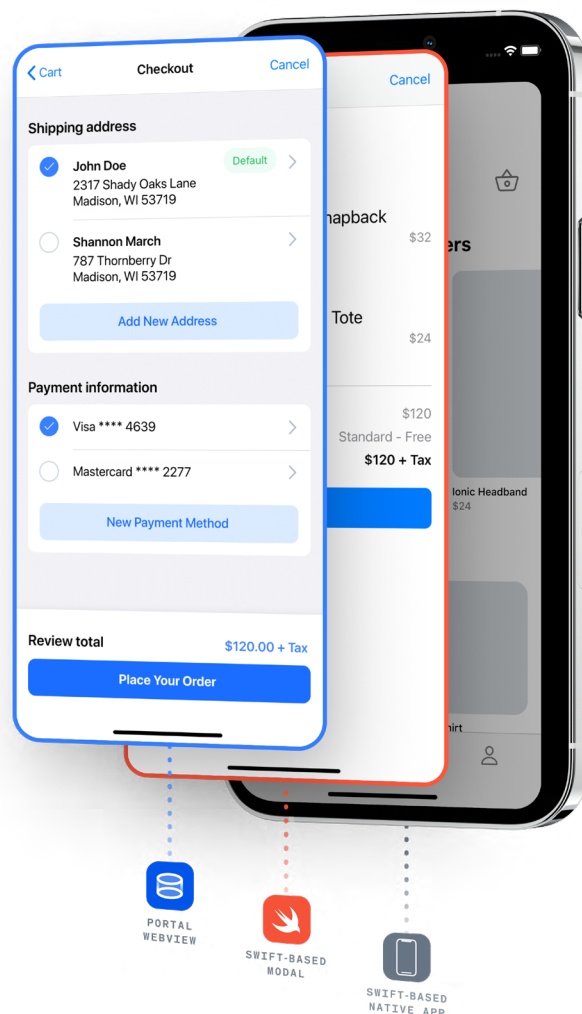
Because the micro frontends themselves are built with the web, it makes the most sense to allow each individual team to write their own end-to-end tests to cover their respective features. Then, focus on simply testing the integration points between the native application and the web-based micro frontend experiences to ensure your app is ready to go.

You're going to need a better Web View.

Development kits from Apple and Google provide stock Web Views for native app developers to start using right away. These out-of-the-box experiences seem like the ideal solution for bringing web and native content together to build and scale mobile apps.

And there are some advantages to doing so:

- Clear separation between features that can be developed independently.
- Immediate deployment with web based features.
- Reuse of web code from mobile app and mobile site.



But there are some disadvantages as well (and the benefits above could come at a cost).

Web Views provided by Apple and Google, can ultimately negatively impact customer experience because of how difficult they are to implement. Creating a seamless experience between the native app and Web Views are extremely important, and stock Web Views are simply lacking.

Why are stock Web Views lacking?

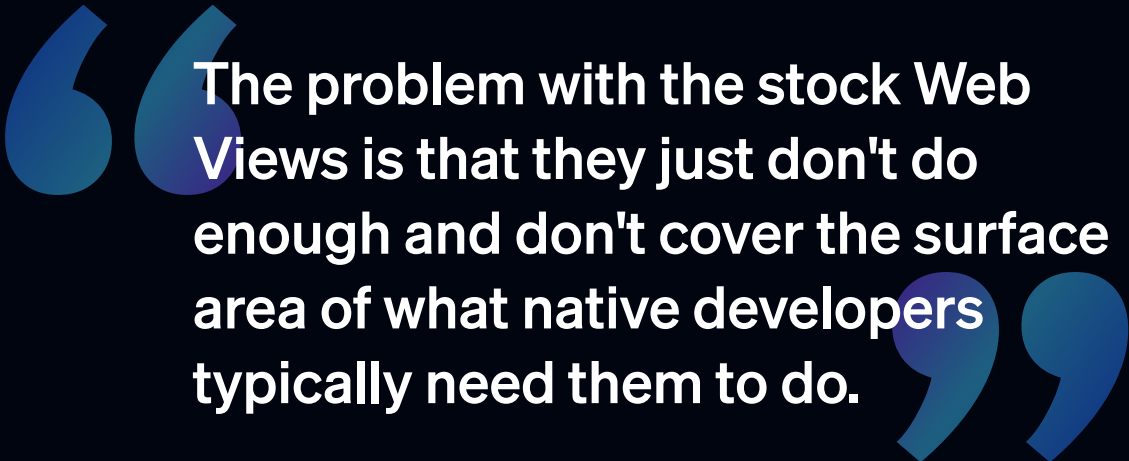
For Web Views to be effective as a supplemental experience for native apps developers, there is a lot of added work that the developers need to do before the Web Views become useful.

Take authentication, for example. In order to log into an account on the native layer then jump to the Web View to see the profile, the system will need to pass that authentication token over to the Web View in order to provide a seamless experience. This kind of communication bridge needs to be created by your development team.

Stock Web Views fall short in a variety of situations where the web and native layers need to interact. For example, native developers must build interactive experiences with native elements such as alert dialogs, keyboard, status bar, scroll regions, all while correctly handling orientation changes. Developers must also maintain general housekeeping like handling delegate methods for navigation, load, errors, and permission requests.

To truly make use of these micro frontends in multiple operating systems, this code needs to be replicated on iOS and Android. Developers must also learn two different APIs on two different operating systems—iOS WKWebView and Android WebView.

The biggest issue with Web Views is perception and consumer experience. It's common for the web features to feel like they don't belong in the rest of the native application, requiring network requests to load web code creates a subpar user experience when compared to the rest of the application.



The problem with the stock Web Views is that they just don't do enough and don't cover the surface area of what native developers typically need them to do.

Max Lynch,
Co-founder and CEO of Ionic

Portals for traditional native and React Native applications

Ionic Portals can help native applications avoid the traditionally messy parts of working with Web Views.



Portals offers a scalability solution for native mobile apps.

With Portals, teams can build and test new and updated features, and they can update these features autonomously rather than waiting to bundle the app for deployment. Teams can then reuse these micro experiences across iOS and Android platforms, with no need to build each feature separately.

Because micro frontends are built using web technologies, native teams can work on important native features while web teams can tackle the web-based features. That way each team handles their own codebases and plays to their strengths.



Portals improves developer efficiency.

Much like the size of the team, the size of the application is a major consideration when using Portals. With managed access to native device features, native teams can focus on their own features and only provide feature access to the web teams as needed, avoiding potentially clunky overlap.

Portals fully supports any stack with JavaScript compatibility, which makes building with it future-proof. With access to a large library of core plugins and over 5 million web developers using and contributing to community plugins, developers can find exactly what they need to build the best mobile applications using Portals.



Portals allows for a better customer experience.

With Portals, teams can automate caching and delivery of their micro frontends. In other words, any updates to the application are delivered automatically to the customer without needing to download a fully packaged and updated binary. Rather, when the customer opens the application, the app automatically checks for any updates in the background and delivers these updates to the customer.

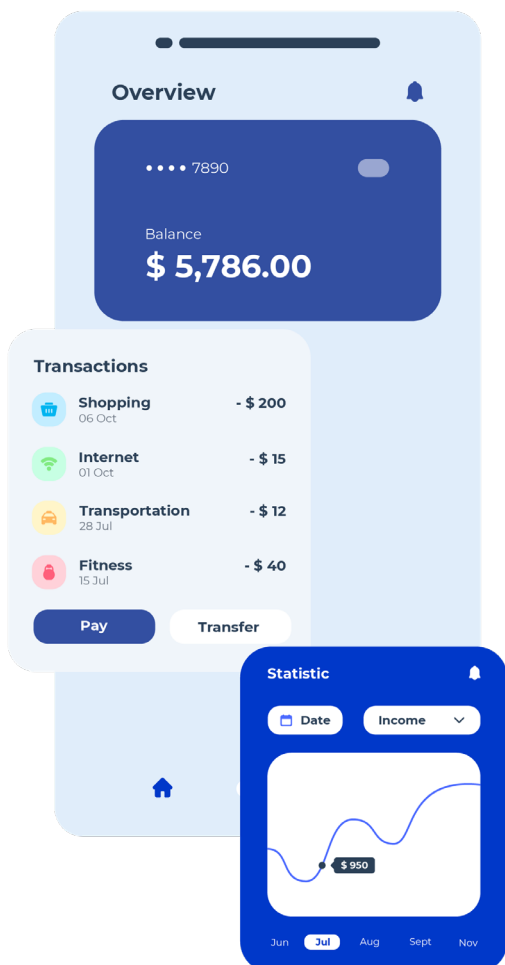
With automated caching and delivery, features can be tested and, if necessary, rolled back instantly. Furthermore, fixes can be deployed instantly to provide a seamless and bug-free customer experience.

Most common use cases

While the uses for micro frontends vary widely across different organizations, there are a few common use cases we've seen.

1. Increase feature velocity by building and shipping in parallel.

Micro frontend architecture enables development teams to add new features and deploy critical bug fixes by working in parallel. This gives teams full autonomy to update, build, and ship features on their own terms, in their own time. Increased feature velocity reduces feature lag between various platforms, namely web, iOS, and Android. By having a consistent feature set across platforms, your customers win.



Use case: A financial institution's solution to combining two apps.

One financial services company sought to integrate two existing Angular apps into an existing native mobile application in order to reduce feature lag between all platforms. Typically, this company saw a 6-month lag between features being available on their web app to being available on their mobile apps.

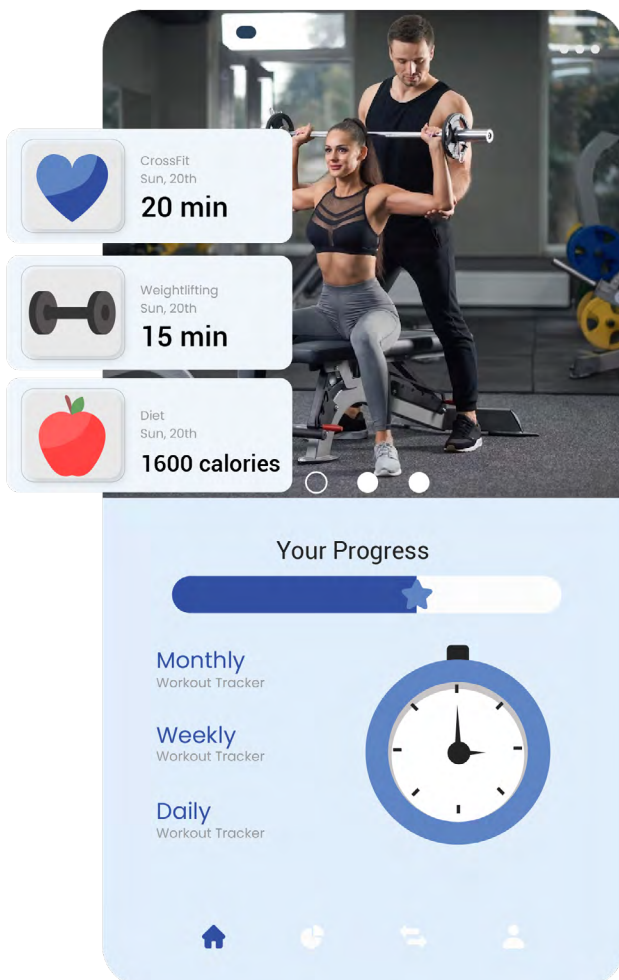
With micro frontend architecture, this company can still use the Angular codebase for their web experience while simultaneously bringing them over into the native application. This allows Angular developers to maintain granular control over their features while native teams work to keep the shell of the native application running.

The end result? Mobile features no longer took 6 months but instead just 6 weeks. The 75% reduction in time to market not only improved the efficiency of their development teams, but also provided a consistent customer experience across all platforms which improved their mobile app ratings.

2. Securely maintain core features when involving different teams.

Parallel app development seems like the right solution for shipping apps and features faster, but what about security? Can web and native development accidentally overstep and cause greater issues when working on various code bases at once?

With micro frontends, mobile native teams can grant multiple internal teams permission to safely and securely contribute to different parts of the app. Limits can be placed on what the web team can access, giving them a sandbox-like experience without the risk of breaking the native functionality or introducing security risks.



Use case: A white-label app brings unique experiences to customers.

This white-label service provides hardware and software to fitness centers around the world. A huge portion of their business is reliant on a branded member mobile app for gyms that purchase their software, ranging from large gyms to smaller studios.

Micro frontends enable third party companies—like those providing payment solutions and rewards offerings—access to develop web-based pieces of content that can be injected into the main mobile experience without interfering with the rest of the code. This specific access allows for parallel development without compromising security. Using micro frontends also resulted in a more consistent customer experience across all platforms which improved their mobile app ratings.

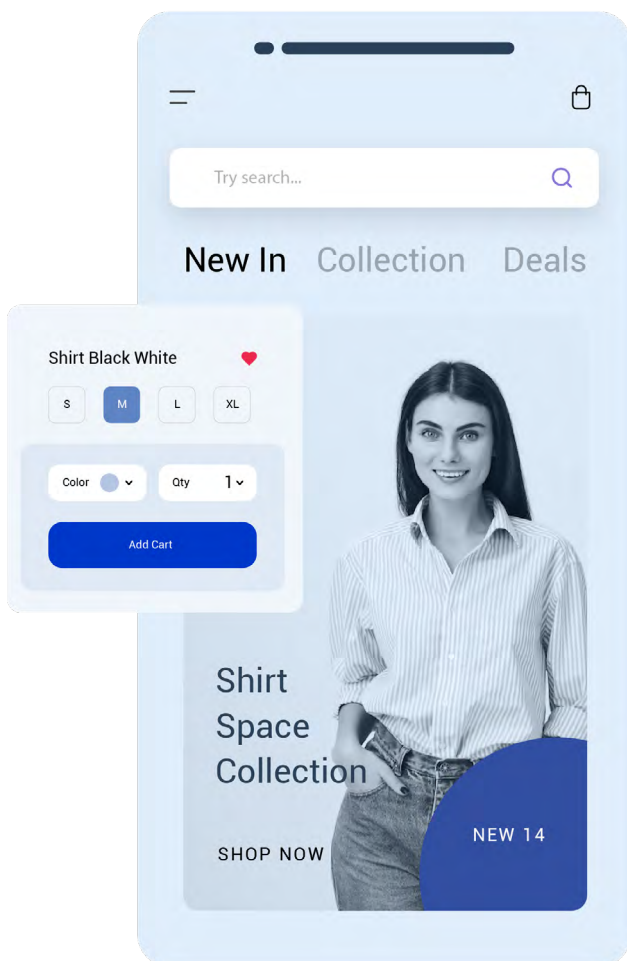
3. Reuse microapps across multiple applications.

Micro frontend architecture not only allows for parallel development across web and native mobile teams, but it enables reuse of web content across multiple platforms.

Web teams developing common experiences like chat functionality, shopping cart checkout, and standard copy like FAQs or legal boilerplate text can reuse experiences across various platforms, rather than just on the web. That means faster time-to-market by eliminating the need to recreate web experiences for each device. It also means less resources are used rewriting code and re-building features that already exist elsewhere.

Reusing content also ensures standardization among different teams for continuity and feature parity. This ultimately leads to a seamless and trusted customer experience.

Additionally, beyond reusing content on native applications, this functionality can expand to smart TVs, refrigerators, car systems, and other platforms.



Use case: An ecommerce platform improves time to market.

An ecommerce company has existing iOS and Android apps and a robust, full-featured web shop. By migrating to a micro frontend architecture, the company can integrate the existing web shop into the iOS and Android applications without rewriting any code, replicating the catalog for each device, or maintaining multiple unique code bases. Instead, the iOS and Android teams simply maintain the native shells while the web team provides the content.

In addition to the online product catalog, the web team can develop the checkout sequence and make updates to other features autonomously without the input of the native teams.

Summary

If you have a large team working on a sizable app, then micro frontends may be for you. Micro frontend architecture enables teams to work in parallel and reuse features that have already been built in their web or native applications. That means that applications can be built and updated faster than ever, while using less resources and still providing a great user experience.

However, there are considerations to adopting this type of architecture. Smaller teams and those with simple apps likely won't realize the full benefit of micro frontends, but it is still worth considering as they scale.

If you're considering micro frontends for your native and React Native applications, then Ionic Portals is a great solution. It empowers your team to work together efficiently, while ultimately putting out a better product faster than ever.



For general questions,
comments, or feedback,
please contact demo@ionic.io.

 [@ionicframework](https://twitter.com/ionicframework)

 [LinkedIn](#)